VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

# Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Marek Svatoš

Bakalářská práce

Vedoucí práce: RNDr. Eliška Ochodková, Ph.D.

Ostrava, 2021

## Abstrakt

Tato bakalářská práce se věnuje průběhu absolvování individuální odborné praxe ve firmě ATLAS consulting spol. s r.o., kde jsem pracoval jako člen backendového týmu. Hlavní část se zabývá vývojem a řešením zadaných úkolů s využitím moderních technologií na aplikacích Codexis a Právní poradna. Závěr práce obsahuje zhodnocení nabytých znalostí před praxí, jejich využití a také výčet znalostí získaných v rámci praxe.

## Klíčová slova

Java, web application

## Abstract

This bachelor thesis describes the course of completing individual professional practice in the company ATLAS consulting spol. s r.o., where I worked as a member of backend team. Main part is dedicated to the process of development and solving assigned tasks using modern technologies on applications Codexis and Právní poradna. The conclusion contains evaluation of gained knowledge prior to the practice, its application and also knowledge gained during practice.

## Keywords

Java, web application

## Poděkování

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, zejména pak vedoucí mé odborné práce paní RNDr. Elišce Ochodkové, Ph.D.

# Obsah

# Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| API | – | Application Programming Interface |
| BE | – | Backend |
| FE | – | Frontend |
| RPC | – | Remote procedure call |
| SQL | – | Structured Query Language |
| POM | – | Project Object Model |
| REST | – | Representational State Transfer |
| BOZP | – | Bezpečnost a ochrana zdraví při práci |
| GDPR | – | General Data Protection Regulation |
| DTO | – | Data Transfer Object |
| POJO | – | Plain old Java object |
| HTML | – | Hypertext Markup Language |
| JWT | – | JSON Web Token |

# Seznam obrázků

# Kapitola 1

# Úvod

Tato práce popisuje průběh absolvované odborné praxe. Možnost zapojit se do vývoje softwaru ve firmě už při studiu považuji za velmi přínosnou. Hlavní motivací pro mě byla práce na reálném projektu, programování v týmu a získání cenných praktických zkušeností. Při výběru společnosti jsem kladl důraz na použití moderních technologií, kterým jsem se chtěl věnovat. Proto jsem si pro vykonání odborné praxe zvolil firmu ATLAS consulting spol. s r.o. [1], kde jsem po úspěšném pohovoru nastoupil na pozici backendového Java vývojáře.

V následujících kapitolách krátce představuji firmu a její hlavní produkty, svou pracovní náplň a popisuji použité technologie při vývoji. V kapitole 5 se věnuji zadaným úkolům. První část úkolů se zabývá právním systém Codexis [2], kde jsem převážně přispěl k refactoringu stávající architektury. V druhé části úkolů jsem se podílel na vývoji zcela nové aplikace Právní Poradna. V každém úkolu popisuji jeho zadání, analýzu a implementaci.

V závěru zhodnocuji a shrnuji absolvování odborné praxe. Uvádím, které znalosti získané při studiu jsem byl schopen využít a také ty, které mi scházely. Práci uzavírám výčtem získaných praktických a teoretických znalostí během praxe.

# Kapitola 2

# Firma

## 2.1 Popis firmy ATLAS consulting spol. s r.o.

ATLAS consulting spol. s r.o. [1], člen skupiny ATLAS GROUP, je ryze českou softwarovou společností, která se již od roku 1992 věnuje vývoji právních a manažerských informačních systémů a aplikací. Přináší na trh moderní software postavený na modulárním principu, který umožňuje sestavit individuální řešení dle potřeb a požadavků klientů. Firma neustále aktualizuje a zdokonaluje funkcionality každého systému. Při vývoji i dodávkách produktů stojí v popředí zájmu vždy uživatel, který ocení bezchybný, bezproblémový a intuitivně ovládaný software. Díky dokonalé znalosti českého prostředí dlouhodobě vyvíjí systémy aplikovatelné na širokou škálu používaných hardware a software.

### 2.1.1 Codexis

Codexis [2] je komplexně řešený software navržený pro efektivní práci s informacemi ze všech oblastí práva. Uživatelsky přívětivý software obsahuje legislativu České republiky i legislativu Evropské unie (u všech zákonů je navíc možné najít nejen jejich aktuální znění, ale také znění historické či budoucí), dokumenty z rozhodovací činnosti českých i evropských soudů, odbornou literaturu, články, komentáře odborníků a další užitečné zdroje informací.

Codexis je určen pro právníky a zástupce právnických profesí, finanční manažery, ekonomy, účetní, úředníky, manažery a specialisty, kteří k výkonu své práce potřebují snadný přístup k právním předpisům, zákonům a informacím k určitému tématu (např. personalistika, účetnictví, GDPR a další).

### 2.1.2 Právní poradna

Právní prostor [3] je právní portál, jehož cílovou skupinou jsou nejenom právní profesionálové a zástupci právnických profesí (advokáti, notáři, soudci, firemní právníci, daňoví poradci atd.), ale

všichni, kteří potřebují právní informace, tj. podnikatelé, management firem, zástupci státní správy či samosprávy, představitelé státních institucí či nadšenci práva.

Právní poradna je nejnovější software firmy ATLAS consulting spol. s r.o, který spadá pod Právní prostor. Slouží jako fórum, ve kterém uživatelé mohou položit jakýkoliv dotaz z právní oblasti. Otázky mohou spadat do několika různých kategoriích jako např. BOZP, obchodní právo nebo GDPR.

# Kapitola 3

# Pracovní náplň

Při nástupu do firmy ATLAS consulting spol. s r.o. [1] jsem se připojil k týmu, který se věnoval vývoji právního systému Codexis [2]. Tým se skládal ze dvou backend vývojářů, dvou frontend vývojářů a frontend kodéra. Hlavní náplní mé práce byly refactoring stávající architektury aplikace Codexis a rozšíření o novou funkcionalitu Workspace, což je doplněk nabízející široké možnosti sdílení a vzájemné spolupráce napříč uživateli.

Refactoring zahrnoval rozbití monolitu aplikace Codexis do gRPC [4] mikroslužeb založených na principu Event Sourcingu [5]. Jednotlivé vytvořené mikroslužby bylo třeba zároveň rozšířit o funkcionalitu pro doplněk Workspace. Veškeré dílčí úkoly pokryly celý pátý semestr mého studia.

Právní Poradna je nejnovější aplikace firmy ATLAS consulting spol. s r.o. a speciálně pro ni vznikl tým, kterého jsem se stal součástí na začátku šestého semestru. Počátečním cílem tedy bylo vybudovat aplikaci do její beta verze. Důraz se kladl na správně navrženou architekturu, u které se opět aplikoval princip Event Sourcingu, a dobře použitelné GraphQL API [6].

## 3.1 Průběh a organizace vývoje

### Návrh a analýza

Vedoucí týmu se mnou při zadání úkolu vždy probral danou problematiku a vysvětlil stávající funkcionalitu nebo navrhl nové řešení. Následně jsem přešel k vlastní analýze a implementaci úkolu. Ze začátku pro mě bylo náročné se zorientovat ve velmi rozsáhlé aplikaci jako je Codexis, a proto jsem musel danou problematiku vícekrát konzultovat. Postupem času jsem už byl schopen menší části navrhnout a vypracovat sám.

## Implementace

Při implementaci prvních úkolů byla klíčová komunikace se zkušenými backend vývojáři, kteří mě v případě problémů dokázali správně navést. Často se totiž stane, že se při implementaci narazí na use case, který se nezohlednil v počátečním návrhu, což následně vede k nové iteraci návrhu. Po dokončení implementace jsem odevzdal úkol ke kontrole vedoucímu týmu, který poskytl zpětnou vazbu a upozornil na chyby. Tento proces zaručuje vyšší úroveň a konzistenci kódu.

## Testování

Části úkolů se složitější logikou bylo třeba pokrýt integračními testy, které jsem psal za pomocí knihovny Testcontainers [7]. Zajišťuje se tak správná funkčnost při integraci nových částí aplikace.

## Nasazení

V závěrečné části vývoje jsem aplikaci nasadil do vývojového a produkčního prostředí. Veškeré automatizované nasazování probíhalo do Kubernetu [8], kde aplikace běží replikovaná v dokerizovaném prostředí.

# Kapitola 4

# Použité technologie

## 4.1 Java

Java [9] je nejpoužívanější programovací jazyk a vývojová platforma. Snižuje náklady, zkracuje dobu vývoje, podporuje inovace a zlepšuje aplikační služby. S miliony vývojářů, kteří provozují přes 51 miliard Java Virtual Machine instancí po celém světě, je Java nadále preferovanou vývojovou platformou pro podniky a vývojáře na celém světě.

## 4.2 Apache Maven

Apache Maven [10] je nástroj pro správu, řízení a automatizaci sestavení aplikací. Základním konceptem je POM, což je XML reprezentace Maven projektu. POM obsahuje veškeré potřebné informace o projektu včetně pluginů použitých při sestavení aplikace a závislostech.

## 4.3 Spring Boot

Spring Framework [11] poskytuje komplexní programovací model pro moderní aplikace psané v Javě, které mohou být nasazené v jakémkoliv prostředí. Spring boot [12] je rozšířením Spring frameworku, které výrazně zjednodušuje a akceleruje vývoj aplikací. Zároveň zbavuje vývojáře nutnosti složitého počátečního nastavení aplikace.

## 4.4 GraphQL

GraphQL [6] je dotazovací jazyk pro API, který předvídatelně poskytuje data. Kompletně a srozumitelně popisuje schématem data v API a tím umožňuje klientovi dotazovat přesně ta data, která potřebuje.

## 4.5 gRPC

gRPC [4] je moderní RPC framework. Umožňuje efektivní komunikaci mezi službami, které využívají architekturu mikroslužeb.

## 4.6 Protobuf

Protobuf [13] je jazykově neutrální mechanismus pro serializaci strukturovaných dat. Jednou definovaná data se mohou zapisovat a číst v různých programovacích jazycích.

## 4.7 Apache Kafka

Apache Kafka [14] je platforma pro streamování eventů, která kombinuje tři klíčové funkcionality:

1. Zapisování a čtení streamů eventů včetně kontinuálního importu a exportu dat z jiných systémů.
2. Ukládání streamů eventů trvale a bezpečně na jakoukoliv dobu.
3. Zpracování streamů eventů při jejich vytvoření nebo retrospektivně.

### Event Sourcing

Tradiční řešení pro persistenci entity je uložení jejího aktuálního stavu. Někdy je ale třeba vědět, jak se entita do svého stavu dostala. Při použití Event Sourcingu [5] se ukládají veškeré změny stavu aplikace jako sekvence neměnných eventů. To umožňuje rekonstruovat předchozí stavy aplikace nebo přehrát aplikaci od počátečního stavu až po stav aktuální.

## 4.8 Apache Solr

Apache Solr [15] je platforma pro vyhledávání v textu, indexaci, facetované vyhledávání, filtraci nebo vyhledávání v dokumentech PDF a ODT. Solr běží jako samostatný vyhledávací server a poskytuje API podobné RESTu.

## 4.9 PostgreSQL

PostgreSQL [16] je objektově-relační databázový systém, který používá a rozšiřuje jazyk SQL. Umožňuje vytvářet vlastní datové typy, funkce a je vysoce škálovatelný.

## 4.10 Liquibase

Liquibase [17] je nástroj pro verzování schématu databáze. Základem je XML soubor obsahující veškeré změny v databázi.

## 4.11 Testcontainers

Testcontainers [7] je Java knihovna, která výrazně zjednodušuje psaní integračních testů. Umožňuje vytvoření instancí jakékoliv platformy běžící v Docker [18] kontejneru po dobu testu.

## 4.12 Mockito

Mockito [19] je Java knihovna, která poskytuje možnost mockování (tj. nahrazení objektů) a simulaci jejich chování.

## 4.13 Docker

Docker [18] poskytuje možnost zabalení aplikace a spuštění v izolovaném prostředí tzv. kontejneru. Výhodou je poměrně malá velikost a flexibilita.

## 4.14 Kubernetes

Kubernetes [8] je systém pro automatické nasazování, škálování a spravování aplikací běžících v kontejnerech.

# Kapitola 5

# Řešení zadaných úkolů

## Časová náročnost úkolů

### Codexis

- Kalendář - 10 dnů
- Oblíbené dokumenty - 7 dnů
- Sledované dokumenty - 14 dnů
- Notifikační centrum - 4 dny

### Právní poradna

- Základní funkcionalita Právní poradny - 14 dnů
- Fulltextové vyhledávání - 5 dnů
- Autentizace - 2 dny
- Systém reputace - 7 dnů
- Editace a historie příspěvků - 2 dny

## 5.1 Kalendář

### Zadání

Pro právní systém Codexis [2] je třeba navrhnout nový a lépe použitelný kalendář. Součástí je integrace s doplňkem Workspace, kde se ke každému pracovnímu prostoru přiřadí jeden nebo více kalendářů.

### Analýza a návrh

Kalendář jsme navrhli jako event sourcovanou mikroslužbu spojenou s hlavní aplikací Codexis, která se chová jako API gateway pro FE. Veškerá data spravuje mikroslužba a poskytuje gRPC [4] API hlavní aplikaci. Důraz jsme kladli na správné navržení GraphQL [6] API, ze kterého vyplynulo API mikroslužby.

### Implementace

Pro začátek jsem namockoval GraphQL API, aby FE mohl začít paralelně pracovat. Definoval jsem tedy potřebné GraphQL DTO, které reprezentovaly kalendář, události a denní pohled na událost. Pro zmiňované objekty jsem vytvořil potřebné GraphQL mutace a query. Na výsledné GraphQL schéma se už FE vývojáři mohli napojit a pracovat s mockovanými daty.

Následně jsem se přesunul k vytvoření základu mikroslužby, kde jsem definoval veškeré závislosti a propojení s databází. Navrhl jsem schéma databáze a implementoval logiku datové vrstvy. Pokračoval jsem gRPC rozhraním pro komunikaci mezi mikroslužbou a hlavní aplikací. Definoval jsem protobuf objekty [13], které vstupují a vystupují z metod gRPC API. Ve výpisu zdrojového kódu 5.1 lze vidět objekt *Calendar*, který reprezentuje kalendář, a ve výpisu 5.2 poté jednotlivé metody API mikroslužby. Datovou vrstvu jsem s rozhraním propojil vrstvou servisní. Komunikaci s mikroslužbou zajišťoval klient, kterého jsem vytvořil v hlavní aplikaci. Dále jsem implementoval servisní vrstvu hlavní aplikace, která zpracovávala data a následně je transformovala vzájemně mezi DTO a protobuf objekty. Tuto vrstvu jsem poté napojil na původně mockované GraphQL API.

V posledním kroku jsem novou funkcionalitu propojil s doplňkem Workspace, kterému jsem přiřadil vlastní výchozí kalendář. Pracovní prostor lze duplikovat, proto jsem musel implementovat také duplikaci kalendáře s jeho veškerým obsahem. Složitější funkcionalitu jsem průběžně testoval integračními testy. Hotovou mikroslužbu jsem nasadil do Kubernetu [8] pro vývojové a produkční prostředí.

```
message Calendar{
  string id = 1;
  string name = 2;
  string color = 3;
```

```
  bool isPublic = 4;
  string owner = 5;
}
```

Listing 5.1: Protobuf objekt reprezentující kalendář

```
service CalendarService{
  rpc addEditCalendar(cz.atlascon.grpc.calendar.messages.Calendar) returns (google
      .protobuf.Empty){
  }
  rpc addEditEvent(cz.atlascon.grpc.calendar.messages.CustomEvent) returns (google
      .protobuf.Empty){
  }
  rpc getCalendars(google.protobuf.StringValue) returns (cz.atlascon.grpc.calendar
      .messages.Calendars){
  }
  rpc getCalendarEvent(google.protobuf.StringValue) returns (cz.atlascon.grpc.
      calendar.messages.CustomEvent){
  }
  rpc getCalendarEvents(cz.atlascon.grpc.calendar.messages.EventsRequest) returns
      (cz.atlascon.grpc.calendar.messages.Events){
  }
}
```

Listing 5.2: Definice gRPC API

## 5.2 Oblíbené dokumenty

### Zadání

Právní systém Codexis [2] umožňuje uživateli definovat množinu dokumentů, se kterými často pracuje. Takové dokumenty se označují jako oblíbené a uživatel k nim má rychlý přístup. Implementaci oblíbených dokumentů je třeba přesunout do vlastní mikroslužby v rámci refactoringu architektury hlavní aplikace. Zároveň je nutné stávající funkcionalitu upravit pro doplněk Workspace, kde každý pracovní prostor má své vlastní oblíbené dokumenty.

### Analýza a návrh

Oblíbené dokumenty jsme opět navrhli jako event sourcovanou mikroslužbu. Aktuální řešení jsme zjednodušili a sjednotili s budoucím znovupoužitím mikroslužby pro další kategorizaci dokumentů.

### Implementace

Implementaci jsem začal založením mikroslužby a jejím základním nastavením. Dále jsem navrhl nové schéma databáze a přesunul refactorovanou datovou vrstvu do mikroslužby. Pokračoval jsem protobuf [13] reprezentací objektů a návrhem gPRC [4] API pro komunikaci mezi hlavní aplikací a mikroslužbou. Doplnil jsem servisní vrstvu mikroslužby, která stojí mezi datovou vrstvou a gRPC rozhraním a slouží k transformaci a zpracování dat. Vytvořil jsem klienta mikroslužby v hlavní aplikaci, kterého vidíme ve výpisu zdrojového kódu 5.3. Dalším krokem byla implementace servisní vrstvy v hlavní aplikaci, která předěluje GraphQL [6] API a klienta mikroslužby. Do GraphQL API jsem přidal potřebné parametry pro rozšíření Workspace a patřičně propojil se servisní vrstvou.

Po dokončení přesunu jsem musel implementovat automatický reimport dat do mikroslužby. Veškerá stávající data byla uložená jako eventy v Apache Kafka [14]. Stačilo tedy implementovat reimport službu, která postupně vytahuje stávající eventy týkající se oblíbených dokumentů, transformuje je na nově vytvořené protobuf objekty a ty zapisuje do nového Kafka topicu.

Nakonec jsem pokryl funkcionalitu nově vzniklé mikroslužby integračními testy a následně nasadil do Kubernetu [8] pro vývojové a produkční prostředí.

```
public class DocTagServiceGrpcClient {

    private final ManagedChannel channel;

    public DocTagServiceGrpcClient(final URI docTagServiceUrl) {
        this.channel = ManagedChannelFactory.getManagedChannel(docTagServiceUrl);
    }

    private DocTagServiceGrpc.DocTagServiceBlockingStub getStub() {
        return DocTagServiceGrpc.newBlockingStub(channel);
    }

    public void setFavorite(final SetFavorite setFavorite) {
        Preconditions.checkNotNull(setFavorite);

        this.getStub().setFavorite(setFavorite);
    }
    ...
};
```

Listing 5.3: Implementace klienta mikroslužby oblíbených dokumetů

## 5.3 Sledované dokumenty

### Zadání

Sledované dokumenty upozorňují uživatele na změny znění v jím vybraných dokumentech. Uživatel si může snadno nastavit jednotlivé dokumenty nebo celé oblasti práva ke sledování a systém ho automaticky upozorní vždy, když v daném zdroji dojde ke změnám. Zadáním je přesun stávající funkcionality a rozšíření pro doplněk Workspace do vlastní mikroslužby, která bude uchovávat stavy sledovaných dokumentů uživatelů, změn v dokumentech a notifikací včetně jejich nastavení pro daného uživatele.

### Analýza a návrh

Základní návrh architektury se podobal úkolu Oblíbené dokumenty, tzn. vytvoření event sourcované gRPC mikroslužby. Původní modul sledovaných dokumentů byl poměrně rozsáhlý, proto bylo třeba úkol rozdělit do čtyř dílčích částí:

1. Sledování dokumentů pro uživatele nebo pracovní prostor.
2. Změny ve sledovaných dokumentech.
3. Notifikace sledovaných dokumentů.
4. Uživatelská nastavení notifikací sledovaných dokumentů.

Součástí návrhu nebyl přesun modulu, který se stará o výpočet změn v dokumentech.

### Implementace

Všechny části úkolu jsem shromáždil do jedné mikroslužby. Vývoj mikroslužby a úpravy v hlavní aplikaci probíhaly velmi podobně jako pro Oblíbené dokumenty včetně automatického reimportu dat. Klíčovým krokem bylo správné propojení s modulem, který se stará o výpočet změn v dokumentech. Zásahy do GraphQL [6] API byly opět minimální, v podstatě se jednalo jen o rozšíření o parametr Workspace.

Složitější funkcionalitu mikroslužby jsem pokryl integračními testy. V hlavní aplikaci jsem refaktoroval testy za použití knihovny Mockito [19], díky které se simulovalo chování mikroslužby. Nové sledované dokumenty jsem nasadil do Kubernetu [8] pro vývojové a produkční prostředí.

Ve výpisu zdrojového kódu 5.4 můžeme vidět SQL příkaz vytvoření tabulky pro změny v dokumentech.

```
<changeSet dbms="postgresql" id="2-add-changed-and-notifications" author="svatos">
        <sql>
            ...
            create table doc_changes(
            change_id text,
            docid text,
            change_type text,
            novely_refs text[],
            change_date bigint,
            added_related_doc_refs bytea[],
            full_toc bytea[],
            created_on bigint,
            client_document_id text,
            effective_date bigint,
            PRIMARY KEY (change_id)
            );
            ...
        </sql>
</changeSet>
```

Listing 5.4: SQL příkaz pro vytvoření tabulky pro změny v dokumentech za pomocí knihovny Liquibase [17]

## 5.4 Notifikační centrum

### Zadání

V doplňku Workspace pro právní systém Codexis [2] se uživatelům zasílají notifikace o aktivitě ostatních členů pracovního prostoru. Pro tyto notifikace je třeba vytvořit notifikační centrum, ve kterém si uživatel může zapnout nebo vypnout různé kategorie notifikací.

### Analýza a návrh

Úpravu vyžadovala mikroslužba, která se stará o uživatelské nastavení, a také notifikační mikroslužba. Dalším krokem bylo propojení s hlavní aplikací a rozšíření GraphQL [6] API.

### Implementace

Notifikační kategorie jsem se rozhodl reprezentovat jako enum (viz. výpis zdrojového kódu 5.5), jehož hodnoty jsou množiny jednotlivých typů notifikací. V notifikační mirkoslužbě jsem rozšířil SQL dotazy na notifikace o parametr seznamu notifikačních kategorií a upravil gRPC [4] API. Poté jsem v uživatelské mikroslužbě přidal persistenci uživatelského nastavení notifikací a definoval jeho výchozí hodnoty. Rozšířil jsem servisní vrstvu a přizpůsobil API.

V hlavní aplikaci jsem do části, která se stará o notifikace, přidal novou funkcionalitu klientů obou mikroslužeb a rozšířil servisní vrstvu. Při dotazu na notifikace jsem nejdříve získal nastavení z uživatelské mikroslužby a s tímto nastavením jsem se dotázal notifikační mikroslužby. Poskytnuté notifikace jsem z protobuf objektů transformoval na GraphQL DTO a vrátil je z GraphQL query.

Oběma mikroslužbám jsem dopsal integrační testy, které pokryly novou funkcionalitu. Mikroslužby jsem poté nasadil do vývojového a produkčního prostředí Kubernetu [8].

```
public enum NotificationCategory {
    ...
    WORKSPACE_SETTINGS(Set.of(WorkspaceNameChangedNotification.class,
        WorkspaceRemoveNotification.class, UserWorkspaceInvitationIdNotification.
        class, NewDomainWorkspaceNotification.class)),
    WORKSPACE_MEMBERS(Set.of(DeclinedInvitationNotification.class,
        UserLeaveWorkspaceNotification.class, ChangedUserRoleNotification.class,
        UserRemovedFromWorkspaceNotification.class)),
    WORKSPACE_CALENDAR(Set.of(NewEventNotification.class)),
    HIGHLIGHTS(Set.of(HighlightNotification.class, DeleteHighlightNotification.
        class)),
    ...
    private final Set<Class> types;

    NotificationCategory(final Set<Class> types) {
        this.types = types;
    }
    ...
}
```

Listing 5.5: Enum reprezentující kategorie notifikací

## 5.5 Základní funkcionalita Právní poradny

### Zadání

Společnost ATLAS consulting spol. s r.o. [1] se rozhodla vyvinout nový produkt spojený s aplikací Právní Prostor [3]. Právní poradna bude sloužit jako fórum, kde uživatelé mohou pokládat otázky z právní oblasti. Úkolem je implementovat BE část základní funkcionality aplikace.

### Analýza a návrh

Na týmové poradě jsme se seznámili s konceptem aplikace a grafickým návrhem. Vedoucí týmu s námi probral klíčové funkční prvky a nastínil základní fáze a úkoly vývoje:

- Vytvoření a zobrazení otázky, odpovědi, komentáře a tagu
- Hodnocení příspěvku
- Změna stavu odpovědi
- Aktivita otázky
- Aplikační a emailové notifikace

#### Architektura

Aplikaci Právní poradna jsem psal s využitím principu Event Sourcingu [5] a návrhového vzoru CQRS [20]. Základem tohoto vzoru je různý přístup k zápisu a čtení dat. Zápis nebo mutace dat reprezentuje event, což je protobuf objekt [13], který zapisuji do příslušného Kafka topicu [14]. Na tomto topicu naslouchají mnou definované reducery, kterým jsem přidělil určitou množinu eventů. Reducery pak eventy postupně zpracují a zapíšou záznamy do databáze. Data se poté čtou přímo z databáze.

### Implementace

#### Mock API

Pro začátek jsem založil nový projekt s potřebnými závislostmi a vytvořil pro něj repozitář. Přešel jsem k mockovaní GraphQL [6] API, které jsme společně s týmem navrhli. Definoval jsem potřebné GraphQL DTO, query a mutace a následně namockoval data vracející se z těchto dotazů. Mockované API jsem nasadil do Kubernetu [8], FE část týmu se na něj napojila a mohla začít pracovat.

### Příspěvky

Po namockování GraphQL API jsem se přesunul k vytvoření datové vrstvy aplikace pro příspěvky. Dle API jsem navrhl schéma databáze za pomocí Liquibase [17], definoval doménové objekty a implementoval komunikaci s databází pomocí návrhového vzoru Table Data Gateway [21]. Pokračoval jsem vytvořením protobuf objektů, které reprezentují zápis nebo mutaci dat, a k nim napsal příslušné reducery. Dále jsem implementoval servisní vrstvu, která zapisuje protobuf objekty do Kafka topicu a dotazuje data z datové vrstvy. Ve výpisu zdrojového kódu 5.6 je definice rozhraní servisní vrstvy pro otázky. Posledním krokem bylo propojení této vrstvy s existujícím GraphQL API.

```
public interface QuestionSerivce {

    QuestionIdDto emitQuestion(final String id, final String author, final String
        title, final String body, final List<String> tags);

    void emitDeleteQuestion(final String id);

    List<QuestionSnippetIdDto> getTopVoted(final int offset, final int limit);

    QuestionDto getQuestion(final String id);

    List<QuestionSnippetIdDto> getQuestionsForUser(final String user, final
        MainContentSort sort, final int offset, final int limit);

    int getTotalCount(final String user);

    List<QuestionSnippetIdDto> getQuestions(final List<String> tags, final
        QuestionSort questionSort, final int offset, final int limit);

    List<QuestionSnippetProto> getQuestionSnippetProtos(final SearchSortProto sort
        , final int offset, final int limit)

    QuestionSnippetDto getQuestionSnippet(final String id);

    List<QuestionSnippetIdDto> getQuestionSnippetIds(final List<Question>
        questions);

    QuestionState getQuestionState(final String id);
```

```
    String getAuthor(final String id);

    String getTitle(final String id);

    List<String> getQuestionWithContent(final String id);
}
```

Listing 5.6: Rozhraní servisní vrstvy pro otázky

### Hodnocení příspěvku

Otázkám, odpovědím a komentářům mohou uživatelé dát „to se mi líbí“ nebo „to se mi nelíbí“, podobně jako na sociálních sítích. Pomocí Liquibase jsem přidal novou tabulku v databázi, která tyto informace bude ukládat, a implementoval datovou vrstvu. Vytvořil jsem mutační protobuf objekt s atributy *userId*, *contentId*, *voteType* a *timestamp* a reducer, který tento event zpracoval a zapsal záznam do databáze. Definoval jsem GraphQL mutaci *voteContent* pro hodnocení příspěvku a napojil ji na servisní vrstvu, která zapsala mutační event do Kafky a přiřadila příslušným příspěvkům hodnocení.

### Stav odpovědi

Odpověď může mít tři základní stavy, které jsem reprezentoval jako enum *AnswerState* s hodnotami *Pending*, *Accepted* a *Guaranteed*. Výchozí stav *Pending* jsem přiřadil nové odpovědi vytvořené běžným uživatelem. Pokud autor otázky tuto odpověď považuje za správnou, může ji převést do stavu *Accepted*. Odpověď se stavem *Guaranteed* má právo vytvořit pouze garantovaný právní zástupce aplikace Právní poradna. Rozšířil jsem GraphQL API o potřebné mutace a vytvořil protobuf objekty, které jsem zapsal do Kafky. Poté jsem přidal zpracování objektů v reducerech a rozšířil datovou vrstvu. Ve výpisu zdrojového kódu 5.7 pak můžeme vidět GraphQL objekt *AnswerInput*, který vstupuje do GraphQL mutace *acceptAnswer*.

```
@GraphQLDto("AcceptAnswerInput")
    public static class AcceptAnswerInput {
        private final String answerId;
        private final boolean accepted;

        @JsonCreator
        public AcceptAnswerInput(
        @JsonProperty("answerId") @Nonnull final String answerId,
        @JsonProperty("accepted") final boolean accepted) {
            this.answerId = answerId;
```

```
        this.accepted = accepted;
    }

    public String getAnswerId() {
        return answerId;
    }

    public boolean isAccepted() {
        return accepted;
    }
}
```

Listing 5.7: GraphQL objekt AcceptAnswerInput

### Aktivita otázky

U otázky sledujeme její poslední aktivitu, která poté slouží k řazení při vyhledávání. Do aktivity se počítá přidání nového příspěvku v rámci otázky, změna stavu otázky, nebo jakékoliv hodnocení. Pro aktivitu jsem vytvořil vlastní tabulku v databázi, implementoval datovou vrstvu a v reducerech jsem při zpracování žádaných eventů aktualizoval databázi. Ve výpisu zdrojového kódu 5.8 vidíme metodu reduceru, která zpracovává příchozí eventy.

```
public void reduce(final String topic, final String key, final Any msg) {
    Preconditions.checkNotNull(topic);
    Preconditions.checkNotNull(key);
    Preconditions.checkNotNull(msg);

    if (msg.is(VoteContentProto.class)) {
        final VoteContentProto vote = Unpacker.unpack(msg, VoteContentProto.
            class);

        this.voteRepo.upsert(vote.getContentId(), vote.getUser(), vote.getVote
            (), vote.getTs());
        this.lastActivityRepo.upsert(vote.getContentId(), vote.getTs());

        this.updateSolr(vote);
    }
}
```

Listing 5.8: Ukázka metody reduceru aktivity příspěvků

### Notifikace

Začal jsem definicí typů základních notifikací, pro které jsem vytvořil příslušné protobuf objekty. Navrhl jsem persistenci dat a implementoval datovou vrstvu. Notifikace vznikají při mutaci nebo vytvoření objektu, např. při vytvoření odpovědi o tomto faktu notifikujeme autora otázky. Vytvořil jsem tedy separátní transformaci těchto mutačních eventů z Kafka topicu za pomocí Kafka streamu na patřičné notifikace. Poté jsem implementoval servisní vrstvu, která tyto notifikace zapisovala do notifikačního Kafka topicu a komunikovala s datovou vrstvou při dotazech. Ve výpisu zdrojového kódu 5.10 je SQL dotaz pro získání uživatelů, kterým je třeba zaslat notifikaci. K notifikačním protobuf objektům jsem analogicky vytvořil GraphQL DTO implementující rozhraní *Notification*. Ve výpisu zdrojového kódu 5.9 můžeme vidět metodu, která transformuje notifikace z protobuf objektů na GraphQL DTO. Posledním krokem bylo rozšíření GraphQL API o query, mutace a subscription, který přes web sockety zasílal zprávy o nových notifikacích FE.

K zasílání emailových notifikací jsem využil stávající notifikační službu. Pro zaslání emailu jsem sestavil objekt podle existující šablony pro příslušnou notifikaci. Tento objekt jsem poté poslal POST dotazem na REST endpoint notifikační služby, která následně odeslala email s atributy.

```
public NotificationDto translate(final Notification notification) {
        final Any msg = Any.parseFrom(notification.getData());
        if (msg.is(AnswerAcceptedNotificationProto.class)) {
            final AnswerAcceptedNotificationProto answerAcceptedNotification =
                Unpacker.unpack(msg, AnswerAcceptedNotificationProto.class);
            final String answerId = answerAcceptedNotification.getAnswerId();
            final String questionId = answerAcceptedNotification.getQuestionId
                ();

            return new AnswerAcceptedNotificationDto(notification.getId(),
                notification.getTitle(),
                    notification.getText(), Instant.ofEpochMilli(notification.
                        getCreatedOn()),
                    Instant.ofEpochMilli(notification.getSeenOn()), Instant.
                        ofEpochMilli(notification.getReadOn()),
                    new AnswerIdDto(answerId), new QuestionSnippetIdDto(
                        questionId));
```

Listing 5.9: Transformace notifikací

```
SELECT author
FROM question
WHERE id = ? AND deleted = FALSE
UNION
SELECT author
FROM answer
WHERE question_id = ? AND deleted = FALSE
UNION
SELECT author
FROM comment
WHERE parent_id = ?
   OR parent_id IN (
    SELECT id
    FROM answer
    WHERE question_id = ?
) AND deleted = FALSE
UNION
SELECT user_id
FROM followed_question f
    LEFT JOIN question q ON f.question_id = q.id
WHERE question_id = ? AND q.deleted = FALSE
```

Listing 5.10: SQL dotazu pro získání uživatelů, kterým je třeba zaslat notifikaci

## 5.6 Fulltextové vyhledávání

### Zadání

Aplikace Právní poradna bude poskytovat fulltextové vyhledávání pro otázky a odpovědi uživatelů.

### Analýza a návrh

Pro hledání v textu jsme zvolili technologii Apache Solr [15]. V GraphQL [6] API jsme navrhli query, které přijímá hledanou frázi a vrací zvýrazněné shody v úryvcích textu.

### Implementace

Pro začátek bylo třeba definovat schéma dokumentu, který se bude do Solru indexovat. Schéma reprezentuje klasický XML soubor, do kterého jsem zapsal jednotlivé atributy dokumentu. Dále jsem

určil operace jako jsou převod do malých písmen, odstranění bílých znaků a stematizaci českého jazyka pro indexovaný dokument a hledanou frázi.

Vytvořil jsem klienta pro komunikaci se Solr serverem a definoval POJO objekt se speciálními Solr anotacemi. Tento objekt reprezentoval indexovaný dokument. Poté jsem vytvořil metodu, která tento objekt předá Solr klientovi, a tím jsem dokončil proces indexace.

Následně jsem se přesunul k samotnému vyhledávání, kde jsem musel implementovat filtraci podle kategorie a stavu otázky a zároveň řazení od nejnovějších, nejaktivnějších a nejoblíbenějších otázek. V Solru se vyhledává podle textového řetězce, který má svou specifickou syntaxi. Pro vytvoření tohoto dotazu jsem definoval metodu *buildQuery*, která ho postupně s veškerými parametry sestavila. Parametry se skládaly z offsetu (počáteční index dotazu), limitu (konečný index dotazu), zmíněného řazení a filtrů. Součástí parametrů byla také definice highlighteru, který vytvoří úryvek z textu a v něm vyznačí hledanou frázi speciálním HTML tagem. Vzhledem k tomu, že dokument reprezentuje otázku nebo odpověď, musel jsem také implementovat seskupování výsledků podle otázek. Z návratové hodnoty dotazu jsem vybral seskupené výsledky, ze kterých jsem získal úryvky s vyznačenou frází a celkový počet nalezených výsledků. Ve výpisu zdrojového kódu 5.11 vidíme metodu *search*, která posílá dotaz na Solr a zpracovává vrácené výsledky. Pro dotaz na vyhledání v textu jsem vytvořil GraphQL query, které přijímalo zmíněné parametry a vracelo zpracovaná data.

Funkcionalitu vyhledávání jsem pokryl integračními testy a Solr společně s aplikací nasadil do vývojového prostředí Kubernetu [8].

```
public FulltextSearchResult search(final SearchInput searchInput) {
        final SolrParams q = this.buildQuery(searchInput);
        try {
            final QueryResponse response = this.solr.query(q);
            final List<GroupCommand> values = response.getGroupResponse().getValues
                ();

            final List<FulltextSearchMatch> matches = values
                    .stream()
                    .map(GroupCommand::getValues)
                    .flatMap(Collection::stream)
                    .map(this::getResultsFromGroup)
                    .map(doc -> this.getFulltextSearchMatch(response, doc))
                    .collect(Collectors.toList());

            final int totalCount = values
                    .stream()
                    .map(GroupCommand::getNGroups)
                    .findFirst()
                    .orElse(0);

            return new FulltextSearchResult(0, searchInput.getLimit(), matches,
                totalCount);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

Listing 5.11: Metoda pro vyhledávání

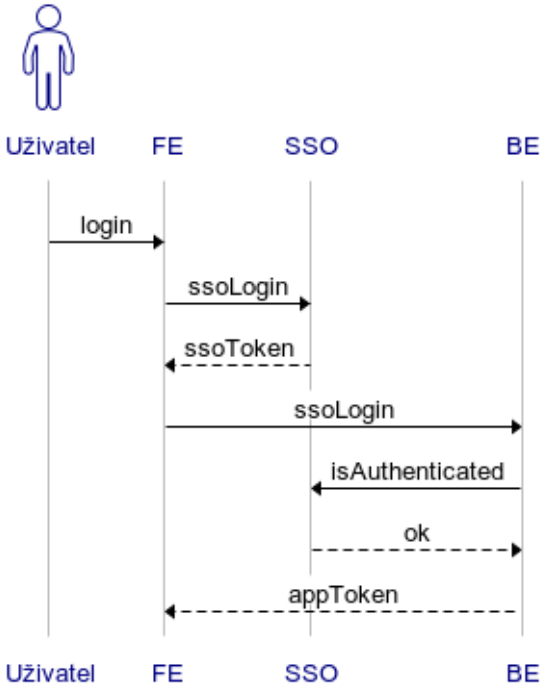
## 5.7 Autentizace

### Zadání

Pro aplikaci Právní poradna je třeba implementovat přihlášení a autentizaci uživatelů.

### Analýza a návrh

Při přihlášení jsme se rozhodli využít stávající SSO službu, která autentizuje uživatele. Takto autentizovanému uživateli se vygeneruje aplikační token, který používá pro přihlášení do aplikace. Na obrázku 5.1 můžeme vidět sekvenční diagram popisující průběh autentizace.




Obrázek 5.1: Sekvenční diagram autentizace uživatele

### Implementace

SSO služba má REST endpoint, který přijímá SSO token a ověřuje jeho platnost. Implementoval jsem tedy POST dotaz na tento endpoint. Následně jsem z SSO tokenu, který má platnost minutu, získal informace uživatele. Z nich jsem vytvořil nový aplikační JWT token s delší platností podepsaný aplikací Právní poradna. Dále jsem přidal GraphQL [6] query, které přijímá SSO token a vrací aplikační token.

Právní poradna poskytuje značnou část funkcí aplikace pouze pro přihlášené uživatele. FE zasílá aplikační token v autorizační hlavičce GraphQL dotazu a BE tak verifikuje uživatele. Při přijetí do-

tazu jsem z přítomného aplikačního tokenu sestavil objekt *User*, který obsahoval atributy uživatele, a dále ho předal do metody zpracovávající příslušný dotaz.

Vytvořil jsem speciální anotaci *Authorized* a do nastavení GraphQL serveru jsem přidal funkcionalitu, která kontrolovala přítomnost objektu *User* se zmíněnou anotací. Pokud má objekt *User* anotaci *Authorized* a je prázdný, vyhodí se speciální výjimka s typem *Unauthorized*. Pro takové případy jsem implementoval GraphQL interceptor, který tuto výjimku zachytí a odpoví na dotaz se statusem 401.

Při prvním přihlášení uživatele jsem se dotázal mikroslužby managementu uživatelů na profilovou fotku uživatele. Získanou fotku jsem společně s ostatními atributy uživatele zapsal do Kafka topicu [14].

## 5.8 Systém reputace

### Zadání

Poradna právního prostoru bude mít systém reputací a oprávnění. Akce každého uživatele ovlivňuje jeho reputaci nebo reputaci ostatních uživatelů. Uživatel má oprávnění na akce podle úrovně své reputace.

### Analýza a návrh

Při specifikovaných akcích uživatele bylo třeba přidat funkcionalitu, která mění jeho reputaci. Reputační hodnoty jsou konfigurovatelná čísla a změna reputace se uloží jako event s okamžitou hodnotou. Reputační akce jsou vratné a získaná reputace se vrací do původního stavu.

### Implementace

Oprávnění uživatele a reputační akce jsem reprezentoval jako hodnoty enumů. Na základě hodnoty reputace se poté uživateli přiřazuje množina oprávnění. Implementoval jsem základy datové a servisní vrstvy, které ukládaly a zpracovávaly reputační eventy.

Poté jsem pokračoval s implementací návratu reputace. Například při smazání jakéhokoliv příspěvku jsem musel shromáždit veškeré reputační eventy pro jednotlivé uživatele týkající se tohoto příspěvku a sečíst jejich hodnoty. Získané číslo jsem převedl do jeho opačné hodnoty, z ní vytvořil nový reputační event a zapsal ho do Kafka [14] topicu.

Jednou z reputačních akcí je nahlášení obsahu příspěvku. Pokud se tak stane několikrát za sebou, dojde k jeho smazání a autorovi se výrazně sníží hodnota reputace. Proto jsem implementoval kontrolu stavu reputace s automatickým zablokováním uživatele a omezením jeho práv v případě, že reputace poklesne pod určitou hranici.

Aby uživatelé měli přehled o stavu reputace, rozšířil jsem funkcionalitu o historii reputačních akcí uživatele, která se zobrazuje v uživatelském profilu. Ve výpisu zdrojového kódu 5.12 je definice GraphQL query *reputationHistory*, které poskytuje historii reputace pro uživatele.

Nakonec jsem přidal kontrolu oprávnění do příslušných GraphQL query a mutací. Nově vzniklou funkcionalitu jsem pokryl integračními testy.

```
@GraphQlQuery("reputationHistory")
public ReputationHistoryOutput reputationHistory(@GraphQLArgument(explode =
    true) final PaginationRequest input, @Authorized final User user) throws
    Exception {
    Preconditions.checkNotNull(input);

    final String userId = user.getId();

    final List<ReputationDto> reputation = this.reputationService.getHistory(
        userId, input.getOffset(), input.getLimit());
    final int totalCount = this.reputationService.getTotalCount(userId);

    return new ReputationHistoryOutput(reputation, input.getOffset(), input.
        getLimit(), totalCount);
}
```

Listing 5.12: GraphQL query historie Reputace

## 5.9 Editace a historie příspěvků

### Zadání

Otázky a odpovědi v Právní poradně budou mít možnost úprav a v případě nevhodného obsahu jsou nahlášeny. Pro tyto příspěvky je také třeba poskytnout historii těchto akcí.

### Analýza a návrh

Pro editaci jsem se rozhodl rozšířit GraphQL [6] mutace pro vytvoření příspěvků. Historii příspěvků pak poskytuje separátní GraphQL query.

### Implementace

Editace příspěvků vyžadovala rozšíření datové vrstvy, ve které jsem vytvořil nové tabulky sloužící k zaznamenávaní změn a implementoval logiku. V GraphQL API jsem sjednotil vytvoření a editaci

příspěvků jako jednu mutaci, kde jedním ze vstupních parametrů je *id* příspěvku. Pokud se toto *id* neposkytne, jedná se o vytvoření nového příspěvku. V opačném případě jde o editaci. Následně jsem upravil servisní vrstvu a napojil na vrstvu datovou.

Nově vzniklé editační tabulky sloužily jako zdroj informací pro historii příspěvků. Pro úplnou historii jsem musel seskupit stavy daného příspěvku v čase s jeho nahlášeními. Reprezentaci této množiny jsem se rozhodl vyřešit pomocí seznamu unionů jako návratové hodnoty GraphQL query pro historii příspěvku. Definici unionu můžeme vidět ve výpisu zdrojového kódu 5.13. Novou funkcionalitu jsem pokryl integračními testy.

```
@GraphQLTypeName("HistoryEvent")
@GraphQLReturnTypes(values = {QuestionHistoryStateDto.class, AnswerHistoryStateDto
    .class, Flag.class})
public class HistoryEventDto extends AbstractGraphQLUnion {
    protected HistoryEventDto(final Object value) {
        super(value);
        Preconditions.checkNotNull(value);
    }

    public static HistoryEventDto of(final Object o) {
        return new HistoryEventDto(o);
    }
}
```

Listing 5.13: Ukázka GraphQL unionu

# Kapitola 6

# Závěr

## 6.1 Uplatněné znalosti

Mezi nejdůležitější znalosti získané v rámci studia bych určitě zařadil algoritmizaci a datové struktury, které se vyučovaly v předmětech Algoritmy I a Algoritmy II. S paradigmatem objektově orientovaného programování, které jsem intenzivně využíval, mě seznámil předmět Programování II. Protože jsem pracoval v jazyce Java, mohl jsem využít znalosti z předmětu Programovací jazyky I, který se právě tomuto jazyku věnoval. K práci s databázemi a jejich návrhu včetně implementace datových vrstev mi pomohly znalosti z předmětů Úvod do databázových systémů a Databázové a informační systémy. Podílel jsem se na vývoji informačního systému, takže návrhové vzory a jejich implementaci jsem mohl zužitkovat z předmětu Vývoj informačních systémů.

## 6.2 Scházející znalosti

Na začátku praxe jsem nejvíce postrádal znalosti moderních technologií, se kterými jsem při studiu bohužel nepřišel do kontaktu. Průběžně jsem se s nimi seznamoval a po absolvování praxe jsem je už dokázal sám používat. Dále mi chyběly znalosti základních principů vývoje moderních aplikací a schopnosti návrhu komplexnějších řešení. Během studia jsem se nesetkal s testováním softwaru ani s kontinuální integrací a nasazením.

## 6.3 Shrnutí

Absolvování odborné praxe považuji za obrovský posun ve svých vývojářských schopnostech zejména díky tomu, že jsem přišel do kontaktu s moderními technologiemi, se kterými jsem předtím neměl žádné zkušenosti. Znalosti získané při studiu jsem mohl aplikovat v praxi na reálných projektech. Podstatným přínosem pro mě bylo také začlenění do týmu a možnost vyvíjet software v kolektivu.

V této bakalářské práci jsem popsal absolvování odborné praxe ve firmě ATLAS consulting spol. s r.o. [1], kde pracuji jako Java developer. Během praxe jsem se podílel na vývoji aplikací Codexis [2] a Právní poradna. Představil jsem firmu, nastínil pracovní náplň, přiblížil technologie použité při vývoji a popsal řešením zadaných úloh.

# Literatura


1. *Atlas Consuling s r.o* [online] [cit. 2021-03-13]. Dostupné z: https://atlasconsulting.cz/.
2. *Codexis* [online] [cit. 2021-03-13]. Dostupné z: https://atlasconsulting.cz/software/codexis/.
3. *Právní Prostor* [online] [cit. 2020-03-11]. Dostupné z: https://www.pravniprostor.cz/.
4. *gRPC* [online] [cit. 2021-03-13]. Dostupné z: https://grpc.io/about/.
5. *Event Sourcing* [online] [cit. 2021-03-13]. Dostupné z: https://martinfowler.com/eaaDev/EventSourcing.html.
6. *GraphQL* [online] [cit. 2021-03-13]. Dostupné z: https://graphql.org/.
7. *Testcontainers* [online] [cit. 2021-03-13]. Dostupné z: https://www.testcontainers.org/.
8. *Kubernetes* [online] [cit. 2020-03-11]. Dostupné z: https://kubernetes.io/.
9. *Java* [online] [cit. 2021-03-13]. Dostupné z: https://www.oracle.com/cz/java/.
10. *Apache Maven* [online] [cit. 2021-03-13]. Dostupné z: https://maven.apache.org/.
11. *Spring* [online] [cit. 2021-03-13]. Dostupné z: https://spring.io/projects/spring-framework.
12. *Spring Boot* [online] [cit. 2021-03-13]. Dostupné z: https://spring.io/projects/spring-boot.
13. *Protobuf* [online] [cit. 2021-03-13]. Dostupné z: https://developers.google.com/protocol-buffers.
14. *Kafka* [online] [cit. 2021-03-13]. Dostupné z: https://kafka.apache.org/intro.
15. *Solr* [online] [cit. 2021-03-13]. Dostupné z: https://en.wikipedia.org/wiki/Apache_Solr.
16. *PostgreSQL* [online] [cit. 2021-03-13]. Dostupné z: https://www.postgresql.org/about/.
17. *Liquibase* [online] [cit. 2021-03-13]. Dostupné z: https://www.liquibase.org/about.
18. *Docker* [online] [cit. 2020-03-11]. Dostupné z: https://www.docker.com/why-docker.
19. *Mockito* [online] [cit. 2020-03-11]. Dostupné z: https://site.mockito.org/.

20. *CQRS* [online] [cit. 2020-03-11]. Dostupné z: `https://martinfowler.com/bliki/CQRS.html`.

21. *Table Data Gateway* [online] [cit. 2020-03-11]. Dostupné z: `https://martinfowler.com/eaaCatalog/tableDataGateway.html`.